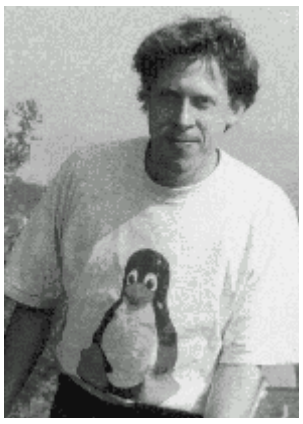
파일 접근 권한



Guido Socher

저자에 대해: 리눅스가 무료이고 리눅스 공동체에 있는 전세계 많은 사람들과 함께 일할 수 있기 때문에 그는 리눅스를 사랑한다. 그는 여유시간에는 그의 여자친구와 보내거나 BBC 라디오 방송을 듣거나 그의 자전거를 타고 교외를 나가거나 리눅스와 노는 것을 즐긴다.

저자에게 편지

요약: 이 기사는 두 부분으로 나눠져 있다:

- 첫 번째인 (기본 파일 접근 권한)은 유닉스 기반에서의 기본적인 파일 접근 권한의 개념을 매우 간단하게 소개이다.
- 두 번째인 (T-bit, SUID와 SGID)는 기본적인 "읽기-쓰기-실행" 플래그를 넘어서는 보다 개선된 기능을 설명이다.

목차:
기본 파일 접근 권한
T-bits, SUID와 SGID

기본 파일 접근 권한

리눅스는 사용자가 다른 접근 권한을 그들이 파일에 정해줄 수 있는 멀티유저 시스템이다. 모든 사용자는 그들의 구분 짓는 사용자 아이디와 고유 번호를 가지고 있다. 또 사용자들은 하나 또는 여럿의 그룹에도 속해있다. 이 그룹들은 몇 몇 사용자 그룹의 접근 제한에 사용되어질 수 있다. 이것의 이점은 여러 작업자들의 팀 작업을 쉽게 한다. 자신의 아이디와 속해 있는 그룹은 'id'명령을 사용한다:

```
>id
uid=550(alice)  gid=100(users)  groups=100(users),6(disk)
```

접근 권한은 각 파일마다 **소유자(user)**, **그룹(group)** 그리고 **다른 사용자(others)**에 대해 기본적인 읽기(**r**), 쓰기(**w**)와 실행(**x**) 권한을 설정할 수 있다. "ls -l" 명령으로 이 권한을 확인할 수 있다.

```
>ls -l   /usr/bin/id
-rwxr-xr-x    1 root   root    8632 May 9 1998 /usr/bin/id
```

파일 "/usr/bin/id"는 사용자가 root이고 root 그룹에 속해 있다.

```
-rwxr-xr-x
```

위의 표시는 파일 접근 권한을 보여준다. 이 파일은 소유자에 대해 읽기 가능(r), 쓰기 가능(w) 그리고 실행 가능(x) 하다. 그룹과 다른 사용자는 읽기(r)와 실행(x)이 가능하다.

각 소유자, 그룹과 다른 사용자의 위치에 있는 3비트의 표시로서 접근 권한을 유추할 수가 있다. **r-x**는 비트 패턴으로의 **101**나 십진수로의 **4+1=5**와 일치한다. r-bit는 십진수 4에 w-bit는 2에 x-bit는 1에 대응된다.

sst 421 (discussed later)	**rwx** 421 user (owner)	**rwx** 421 group	**rwx** 421 others

chmod 명령으로 이런 권한을 바꿀 수가 있다. 보안에 의해 단지 root와 파일의 소유자만이 권한을 변경할 수가 있다. 권한의 숫자 표현이나 기호 표현 모두 chmod에 사용되어질 수 있다. 기호 표현은 [ugoa][+-][rwx] 이다. 이것은 u (user=file owner), g (group), o(others), a(all=u and g and o)와 권한을 더할지 삭제할지를 결정하는 + 또는 - 그리고 r(read) w(write) x(execute)에서 비롯된 권한 기호 표현의 문자로 표현된 것이다. "file.txt"을 모두(**a**ll)에게 쓰기 가능(**w**ritable)하게 하기 위해서는:

```
>chmod a+w    file.txt
or
>chmod 666    file.txt
>ls -l    file.txt
-rw-rw-rw-    1 alice    users    79 Jan 1 16:14 file.txt
```

"chmod 644 file.txt"는 그 파일의 권한을 소유자만이 읽기+쓰기 가능하고 그 외에는 단지 읽기만 가능한 정상의 권한으로 돌려놓는다.

어느 디렉토리로 들어가는 것(cd 명령을 이용하여)은 그 디렉토리를 실행하는 것과 같다. 그래서 디렉토리의 정상 권한은 **755**이지 644가 아니다:

```
>chmod 755   mydir
>ls -ld   mydir
drwxr-xr-x    2 alice    users    1024 Dec 31 22:32 mydir
```

umask는 사용자의 기본 권한을 정의한다. 기본 권한은 새로운 파일(그리고 디렉토리, 등...)들이 생성될 때 적용된다. umask는 사용자가 원하지 않는 곳의 비트를 세트하고 그것을 십진수로 표현한 값을 매개변수로 받는다.

좋은 예로서 "umask 022"가 있다. "022"로 모든 사람들이 파일을 읽을 수 있고 "cd"로는 디렉토리를 변경할 수 있지만 소유자만이 그것을 변경할 수 있다. 현재의 umask가 어떻게 설정되어있는지 알아보기 위해서는 아무 매개 변수 없이 umask를 실행시키면 되다.

아래에 umask와 chmod를 사용한 예이다:

```
umask는 좋은 표준 값이다
>umask
22

사용자의 에디터로 myscripf라는 파일을 만들어라:
>nedit   myscript   (or vi myscript ...)
아래의 내용을 입력하라:

#!/bin/sh
#myscript
echo -n "hello "
whoami
echo "This file ( $0 ) has the following permissions:"
ls -l $0 | cut -f1 -d" "
위 스크립트를 저장하라.

이제 위의 파일은 644의 권한을 가졌다:
>ls -l  myscript
-rw-r--r--   1 alice  users  108 Jan 1 myscript

위의 스크립트를 실행하기 위해서는 실행 가능으로 만들어야 한다:
>chmod 755 myscript
or
>chmod a+x myscript
이제 실행시켜라:
>./myscript
```

일반 컴파일된 바이너리 파일이 단지 실행 가능만 필요하지만 스크립트는 반드시 읽고 실행가능하게 끔 해야한다는 것에 주목하라. 이것은 스크립트가 반드시 쉘과 같은 인터프리터에 의해 반드시 읽혀져야 하기 때문이다. 위의 스크립트를 실행하면 다음과 같이 나온다:

```
hello alice
This file ( ./myscript ) has the following permissions:
-rwxr-xr-x
```

T-bit, SUID와 SGID

한동안 리눅스에서 작업을 하다보면 아마도 "rwx" 비트 이외 보다 많은 파일 권한이 있음을 발견할 것이다. 사용자의 파일 시스템을 둘러보면 "s"와 "t"가 있는 것을 볼 것이다:

```
>ls -ld /usr/bin/crontab  /usr/bin/passwd  /usr/sbin/sendmail  /tmp

drwxrwxrwt   5 root    root    1024 Jan 1 17:21 /tmp
-rwsr-xr-x   1 root    root     0328 May 6 1998 /usr/bin/crontab
-r-sr-xr-x   1 root    bin      5613 Apr 27 1998 /usr/bin/passwd
-rwsr-sr-x   1 root    mail    89524 Dec 3 22:18 /usr/sbin/sendmail
```

이 "s"와 "t" 비트는 무엇일까? 이 권한 비트를 보면 4 * 3 비트 길이이다. 사실 chmod 755는 chmod 0755의 축소된 모습니다.

The t-bit

t-bit(가끔 끈끈이 비트 sticky bit 라고 도 함)는 디렉토리와 사용되어질 때에만이 유용하다. 이것은 위에서 보는 것과 같이 /tmp 디렉토리와 같이 사용되어진다.

일반적으로 (디렉토리에 T-bit 설정 없이) 파일들을 담고 있는 디렉토리가 그 파일들을 지우려고 하는 사람에게 쓰기 가능하게 되어 있다면 그 파일들은 지워질 수 있다. 그래서 사용자가 아무나 파일을 넣어 놓을 수 있는 디렉토리를 가지고 있다면 어느 누구나 다른 사람의 파일을 지울 수 있다.

T-bit는 바로 이 규칙을 바꾼다. T-bit를 설정하면 그 디렉토리의 소유자와 파일의 소유자만이 그 파

일을 지울 수 있다. chmod a+tw나 chmod 1777로 T-bit를 설정할 수 있다. 예를 들면:

```
Alice가 T-bit가 설정된 디렉토리를 만든다:
>mkdir mytmp
chmod 1777 mytmp

Bob이 한 파일을 디렉토리에 넣는다:
>ls -al
drwxrwxrwt   3 alice    users   1024 Jan  1 20:30 ./
-rw-r--r--  1 bob    users      0 Jan  1 20:31 f.txt

이 파일은 Alice (디렉토리 소유자)와 and Bob (파일 소유자)에 의해서 지워질 수 있다. 그러나 Tux
에 의해서는 지워질 수 없다:
>whoami
tux
>rm -f f.txt
rm: f.txt: Operation not permitted
```

S-bit 사용자에게 설정

리눅스에서 프로세스들은 사용자 아이디를 기반으로 실행된다. 이것은 사용자들이 접근할 가능성이 있는 자원(파일 등)에 사용자들이 접근할 수 있게 한다. 실제 사용자 아이디(real user-ID)와 유효 사용자 아이디(effective user-ID)의 2가지 사용자 아이디가 있다. 유효 사용자 아이디가 파일들에 대한 접근을 종료하게 한다. 아래의 스크립트를 idinfo라는 이름으로 저장하고 실행 가능하게 만들어라 (chmod 755 idinfo).

```
#!/bin/sh
#idinfo: Print user information
echo " effective user-ID:"
id -un
echo " real user-ID:"
id -unr
echo " group ID:"
id -gn
```

실행시키면 프로세스가 실행되는 사용자 아이디와 사용자의 그룹 아이디를 볼 수 있을 것이다:

```
 effective user-ID:
alice
 real user-ID:
alice
 group ID:
users
```

Tux가 당신의 idinfo 프로그램을 실행하면 Tux의 아이디로 돌아가는 프로세스를 보여주는 비슷한 결과를 얻을 것이다. 이 프로그램의 결과는 파일을 소유한 사람에 영향을 받는 것이 아니라 파일을 실행한 사람에 의에 영향을 받는다.

보안의 이유로 S-bit는 스크립트들(perl 스크립트는 제외)에서는 작동이 안되고 단지 바이너리(컴파일된 코드)에만 작동된다. 그러므로 우리는 idinfo 프로그램을 C 프로그램으로 만들어야 한다:

```
/*suidtest.c*/
#include <stdio.h>
#include <unistd.h>
int main(){
/*secure SUID programs MUST
*not trust any user input or environment variable!! */

char *env[]={"PATH=/bin:/usr/bin",NULL};
char prog[]="/home/alice/idinfo";
if (access(prog,X_OK)){
    fprintf(stderr,"ERROR: %s not executable₩n",prog);
    exit(1);
}
printf("running now %s ...₩n",prog);
execle(prog,(const char*)NULL,env);
perror("suidtest");

return(1);
}
```

위의 프로그램을 "gcc -o suidtest -Wall suidtest.c"로 컴파일하고 S-bit를 사용자에게 설정하라:

```
>chmod 4755   suidtest
or
>chmod u+s   suidtest
```

위의 프로그램을 실행하라! 어떤 일이 일어났는가? 아무 일도 안 일어났나? 다른 사용자로 이 프로그램을 실행하라!

suidtest 파일은 alice 의해 소유되어져 있고, 정상적으로 파일 소유자를 위해 설정되어져야 할 곳에 x가 설정된 S-bit를 가지고 있다. 이것은 그 파일을 실행하는 사용자보다는 파일을 소유하고 있는 사용자의 유효 사용자 아이디에 의해 실행되게끔 한다. 만일 Tux가 이 프로그램을 실행시키면 다음과 같이 된다:

```
>ls -l suidtest
-rwsr-xr-x   1 alice   users   4741 Jan 1 21:53 suidtest
>whoami
tux

running now /home/alice/idinfo ...
effective user-ID:
alice
real user-ID:
tux
group ID:
users
```

보다시피 root가 S-bit를 가지고 있는 파일을 소유하고 있을 때 특별히 아주 강력한 기능이다. 어느 사용자도 단지 root만이 할 수 있는 일을 할 수 있다. 보안을 조금 생각하고 얘기하자면, 사용자가 SUID 프로그램을 작성할 때에는 사용자가 꼭 의도하는 데로 사용되어질 수 있는지 반드시 확인해야 한다. 환경변수나 환경 변수를 사용하는 함수에 의존하지 말고 항상 코드 상에서 확실하게 경로를 설정해야 한다. 결코 사용자의 입력(config 파일, command line 매개 변수 ...) 을 믿지 마라. 사용자의 입력을 바이트마다 검사하고 당신이 유효하다고 생각하는 값과 비교해라.

SUID 프로그램이 root에 의해 소유되어 있으면 유효 사용자 아이디와 실제 사용자 아이디 모두가 (setreuid() 함수를 사용하여) 설정되어질 수 있다

Set-UID 프로그램들은 흔히 root에 의해 일반적으로는 root만이 할 수 있는 일을 일반 사용자들이 할 수 있게 하는데 사용되어진다. root로서 당신은 당신의 시스템에 있는 어떤 사용자나 당신의 ppp-on/ppp-off 스크립트를 실행할 수 있게 suidtest.c 를 변경할 수 있다.

S-bit 그룹에 설정

S-bit가 그룹에 대해 설정된 실행 가능한 파일은 사용자의 그룹 아이디 기반에서 실행된다. 이것은 위에서 설명한 사용자에 대한 S-bit와 매우 유사하다.

S-bit가 한 디렉토리의 그룹에 대해 설정되어 있다면 그 설정은 그 디렉토리에서 만들어지는 모든 파일에 설정이 된다. Alice는 2개의 그룹에 속해 있다:

```
>id
uid=550(alice)  gid=100(users)  groups=100(users),6(disk)
```

일반적으로 그녀가 만드는 파일들의 그룹은 users로 설정된다. 그러나 디렉토리가 disk 그룹으로 만들어져 있고 그 그룹에 대해 S-bit가 설정이 되어 있다면 만들어 지는 모든 파일들 역시 disk 그룹 아이디를 가진다:

```
>chmod 2775 .
>ls -ld .
drwxrwsr-x  3 tux   disk     1024 Jan 1 23:02 .

만일 Alice가 지금부터 새로운 파일을 이 디렉토리에서 만든다면 그 파일들의 그룹은 모두 disk로 설정이 될 것이다.
>touch newfile
>ls -l newfile
-rw-r--r--   1 alice    disk      0 Jan 1 23:02 newfile
```

이 기능은 당신이 어느 팀에서 여러 사람들과 같이 작업을 하고 그 팀이 작업하는 디렉토리 안의 파일들에는 모두 올바른 그룹이 설정될 수 있게 하는 좋은 기능이다. 특히 그룹 이외 사람들의 접근을 막을 수 있게 하는 027 umask가 사용자들에게 설정되어 있는 환경에서 유용하다.

번역 : 채희주